

Práctica de Oracle: Aspectos Teóricos

Tabla de Contenidos

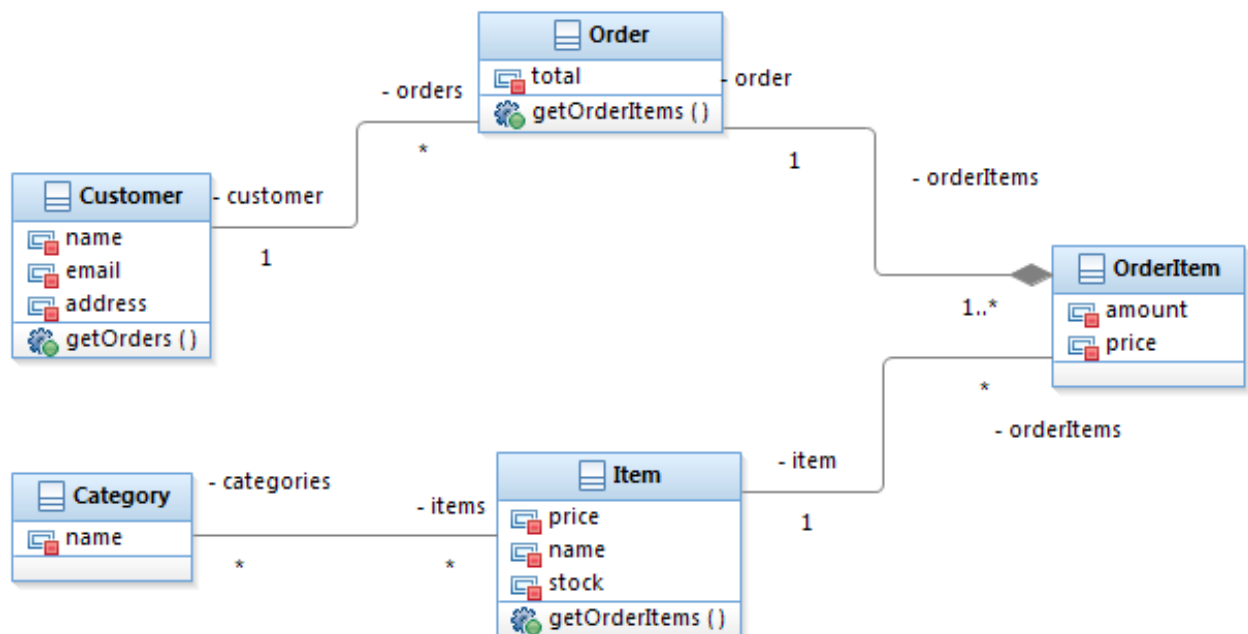
1. Creación de tipos.....	2
1.1. Creando clientes	3
1.2. Creando productos.....	4
1.3. Creando categorías.....	4
1.4. Creando elementos de orden.....	5
1.5. Creando órdenes	5
1.6. Actualizando tipos	6
1.7. Agregando funciones.....	7
Implementando getOrders para el objeto Customer	7
Implementando getOrderItems para el objeto Order.....	8
Implementando getOrderItems para el objeto Item	8
2. Ejemplos de inserción de datos	10
2.1. Inserción de items.....	10
2.2. Inserción de clientes.....	10
2.3. Poniendo al ítem de un OrderItem.....	10
2.4. Creando órdenes.....	10
2.5. Asignando a una categoría un conjunto de productos	10
3. Consultas.....	12
Script de Consultas	12
3.1. Mostrar las órdenes	12
3.2. Mostrar la categoría a la que pertenece un ítem.....	12
3.3. Mostrar los ítems de una orden	13
3.4. Items que no estén ligados a una categoría	13
3.5. Nombre del cliente en órdenes con más de 3 productos diferentes comprados	14
3.6. Órdenes de un cliente y con un ítem determinado	14
III. Fuentes de consulta	15

1. Creación de tipos

Para explicar cómo funciona Oracle orientado a objetos, lo explicaremos basándonos en un modelo sencillo de una tienda virtual. Consideraremos en esta tienda un alcance básico, que contendrá los siguientes elementos:

- Clientes. Que hacen las compras. Serán llamados Customers
- Órdenes. Las órdenes de compras que se dan de alta cuando un cliente hace una compra. Serán llamadas Orders
- Elementos de orden. Una orden puede tener muchos elementos de orden. Un elemento de orden está compuesto de un producto y una cantidad. Serán llamados OrderItems
- Productos. Los que será comprados por el cliente. Serán llamados Items
- Categorías. A las que está asociada un producto. Un producto puede estar asociado a varias categorías, y viceversa.

Una representación gráfica quedaría representada de la siguiente forma:



1.1. Creando clientes

```
/* Creando un tipo de dato */  
CREATE TYPE addressUdt AS OBJECT (  
    street varchar(30),  
    exterior number(8),  
    zip varchar(5)  
) NOT FINAL;
```

CREATE TYPE type name es la instrucción de inicio básica para crear un tipo de dato en Oracle. En este caso, se nombró al tipo **addressUdt**. La instrucción clave **AS OBJECT** es necesaria también para crear el tipo de dato.

Posteriormente se abren paréntesis, y dentro de los mismos se definen los atributos del tipo, de la misma manera en que se hace para la creación de una tabla. Finalmente, la palabra clave **INSTANTIABLE** sirve para indicar que se podrán crear instancias de este objeto. Es el comportamiento por defecto, así que para nuestros intereses podríamos haberlo omitido. Por otro lado, la instrucción clave **NOT FINAL** indica que se pueden crear subtipos de este tipo. Puesto que la instrucción por defecto es lo opuesto (**FINAL**), era necesario indicar este comportamiento de forma explícita.

```
/* Creando tipo de dato customerUdt */  
  
CREATE TYPE customerUdt AS OBJECT(  
    pid varchar(11),  
    name varchar(20),  
    email varchar(20),  
    address addressUdt  
) INSTANTIABLE NOT FINAL;
```

Nótese que creamos el cliente y una de sus atributos es el tipo de dato **addressUdt** que definimos anteriormente.

¡Listo!, ahora podemos empezar a crear tablas. Es importante recordar que, a fin de cuentas, los *types* son una abstracción de Oracle que nos permite hacer el modelado de un sistema en el nivel de base de datos; sin embargo, seguiremos usando las tablas convencionales para guardar nuestros datos. De este modo, para crear la tabla del objeto **Customer** usaremos el tipo **customerUdt** con ayuda de la keyword **OF**, misma que se usa para definir tablas de tipo objeto.

Haremos 2 cosas más:

- Señalaremos al atributo *pid* como llave primaria
- Haremos que el identificador del objeto sea generado por el sistema con la instrucción **OBJECT IDENTIFIER IS SYSTEM GENERATED**. Esto sirve para que, en caso este objeto sea referenciado por otra tabla, se use un identificador único – oculto dentro de la tabla – que haga las veces de llave foránea (visitar

http://download.oracle.com/docs/cd/B28359_01/appdev.111/b28371/adobjdes.htm#i443361
para más información)

```
/* Creando la tabla customer en base al tipo customerUdt */
CREATE TABLE customer OF customerUdt(
    pid PRIMARY KEY
) OBJECT IDENTIFIER IS SYSTEM GENERATED;
```

1.2. Creando productos

```
/* Creando tipo de dato itemUdt */
CREATE TYPE itemUdt AS OBJECT (
    pid varchar(11),
    price number(10,2),
    name varchar(20),
    stock number(6)
) NOT FINAL;
```

Ya que el generado del id de cada tabla la genera el sistema por defecto, en las siguientes instrucciones para crear el resto de objetos no usaremos la instrucción **OBJECT IDENTIFIER IS SYSTEM GENERATED**.

```
/* Creando table item en base al tipo itemUdt*/
CREATE TABLE item OF itemUdt(
    pid PRIMARY KEY
);
```

1.3. Creando categorías

Ahora creamos un tipo de dato que nos permite hacer la relación *items*, misma que representa un conjunto de productos asociados con una categoría. Para ello, necesitamos crear un arreglo, mismo que haremos con la instrucción **AS VARRAY(varray_size)**. Este tipo de dato debe tener un tamaño fijo y un tipo de dato. Este último será, efectivamente, de tipo *itemUdt*, sólo que usaremos una referencia a este tipo en lugar de hacer la asociación de manera directa con ayuda de la palabra clave **REF**.

```
/* Creando arreglo itemUdtArr en base al tipo itemUdt */
CREATE TYPE itemUdtArr AS VARRAY(50) OF REF itemUdt;
```

Ya con lo anterior, podemos proceder a crear la categoría.

```
/* Creando el tipo de dato categoryUdt */
CREATE TYPE categoryUdt AS OBJECT (
    pid number(8),
    name varchar(20),
```

```

        items itemUdtArr
    ) NOT FINAL;

```

Y también la tabla correspondiente a la categoría:

```

/* Creando la tabla category en base al tipo categoryUdt*/
CREATE TABLE category OF categoryUdt (
    pid PRIMARY KEY
) OBJECT IDENTIFIER IS SYSTEM GENERATED;

```

1.4. Creando elementos de orden

Ahora crearemos el tipo de dato para la relación entre órdenes y productos. Nuevamente, en caso de la relación que tiene con los ítems, usaremos la palabra reservada REF, como explicamos en la parte superior:

```

/* Creando tipo de dato orderItemUdt */
CREATE TYPE orderItemUdt AS OBJECT(
    pid varchar(11),
    price number(10,2),
    amount number(3),
    relatedItem REF itemUdt
) NOT FINAL;

```

1.5. Creando órdenes

A continuación crearemos el tipo de dato **orderUdt**. Para esto, además de las propiedades que contiene, será necesario asignar 1 atributo extra que se encuentra en sus relaciones: *customer* (para lo que usaremos el tipo **customerUdt**):

```

/* Creando tipo de dato orderUdt */
CREATE TYPE orderUdt as OBJECT(
    pid number,
    total number(10,2),
    buyer REF customerUdt
) INSTANTIABLE NOT FINAL;

```

Posteriormente, crearemos la tabla del objeto **Order** en base al tipo **orderUdt**, usando para sus atributos tanto una restricción referente fuera de línea (las que inician con la palabra clave **CONSTRAINT** como una restricción referente en línea (en este caso, usando la instrucción clave **SCOPE IS**).

```

/* Creando la tabla order en base al tipo orderUdt*/
CREATE TABLE buy_order OF orderUdt (

```

```

        CONSTRAINT buy_orderPK PRIMARY KEY (pid),
        buyer SCOPE IS customer
    ) OBJECT IDENTIFIER IS SYSTEM GENERATED;

```

1.6. Actualizando tipos

Ahora bien, notamos que al objeto **orderItemUdt** le falta la relación con el objeto **Order**. Así, usaremos la siguiente cláusula para agregar la relación:

```

ALTER TYPE orderItemUdt ADD ATTRIBUTE (
    relatedOrder REF orderUdt
) CASCADE;

```

No nos olvidemos de crear la tabla para el objeto **orderItemUdt**:

```

/* Creando la tabla orderItemUdt en base al tipo orderItemUdt */
CREATE TABLE order_item OF orderItemUdt (
    CONSTRAINT orderItemPK PRIMARY KEY (pid),
    relatedItem SCOPE IS item,
    relatedOrder SCOPE IS buy_order
) OBJECT IDENTIFIER IS SYSTEM GENERATED;

```

También podemos ver que al tipo de dato **itemUdt** le falta la relación con el objeto **Category**. Para poder crear esta relación, no obstante, hay que crear un VARRAY de tipo **categoryUdt**.

```

/* Creando arreglo categoryUdtArr en base al tipo categoryUdt */
CREATE TYPE categoryUdtArr AS VARRAY(50) OF REF categoryUdt ;

```

Ahora sí, podemos hacer la asociación:

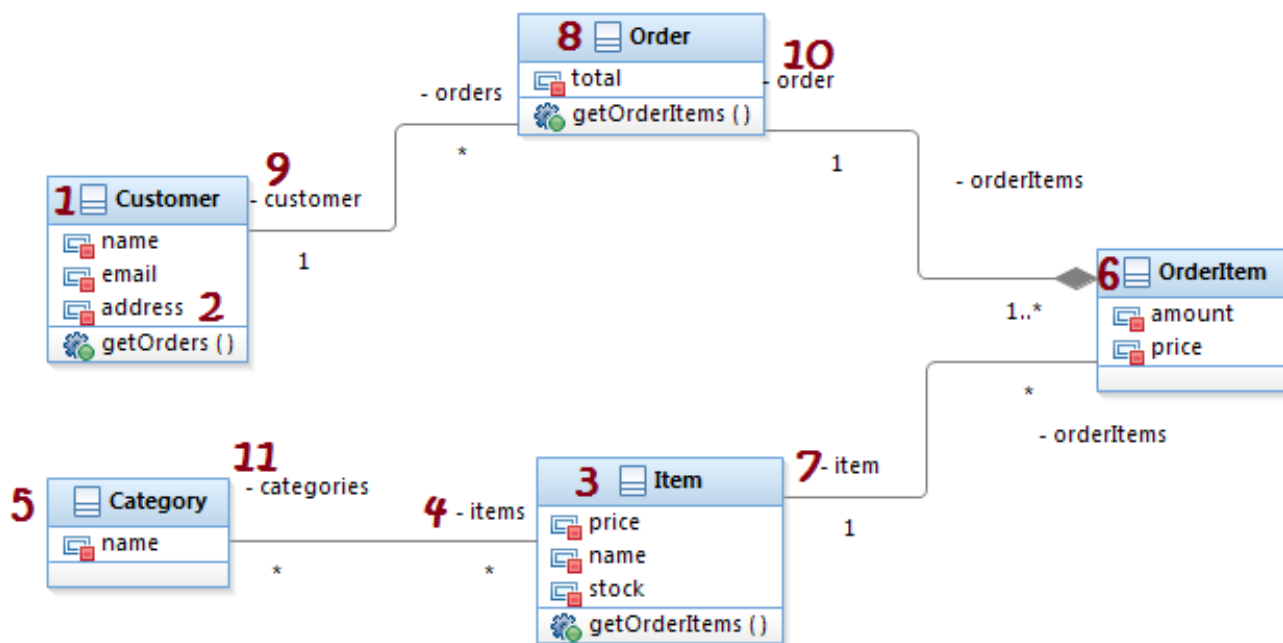
```

ALTER TYPE itemUdt ADD ATTRIBUTE (
    categories categoryUdtArr
) CASCADE;

```

1.7. Agregando funciones

Ahora bien, si examinamos el diagrama inicial, podremos ver que hemos seguido el siguiente orden para establecer las relaciones entre objetos:



Aquí es el momento de explicar lo siguiente: Los arreglos de objetos son asociados directamente a un objeto cuando representan una relación muchos-a-muchos. Así fue como lo hicimos entre el objeto **Category** y el objeto **Item**, por ejemplo.

Ahora bien, ¿qué sucede cuando un objeto tiene una relación de 1-a-muchos con otro? En este caso, usaremos la instrucción clave **MEMBER FUNCTION** en Oracle. En nuestro diagrama, representamos la necesidad de estas funciones en cada objeto en sintaxis, justamente, de métodos.

Implementando `getOrders` para el objeto **Customer**

Empezaremos a definir la función `getOrders` para el objeto **Customer**. Así, necesitaremos crear un arreglo que contendrá el resultado que arrojará la función:

```
/* Creando arreglo orderUdtForCustomerArr en base al tipo orderUdt */  
create type orderUdtForCustomerArr as VARRAY(1000) of REF orderUdt;
```

Posteriormente, alteraremos el objeto **Customer** para agregar esta función

```
/* Alterando el tipo de dato customerUdt */  
ALTER TYPE customerUdt ADD MEMBER FUNCTION getOrders(arg1 in Number) RETURN  
orderUdtForCustomerArr CASCADE;
```

Una vez hecho lo anterior, necesitamos decirle a Oracle cómo queremos que esa función sea implementada. Y aquí empezaremos a entender cómo se implementan las funciones de un tipo de dato. Para poder hacer lo anterior con la función *getOrders*, necesitaremos de la instrucción clave **CREATE OR REPLACE TYPE BODY**, puesto que esto nos permite implementar el código de los métodos que declaramos en el tipo.

```
/* Implementando la función getOrders para el objeto Customer */
CREATE OR REPLACE TYPE BODY customerUdt AS
    MEMBER FUNCTION getOrders(arg1 in Number) RETURN orderUdtForCustomerArr IS
        cosa orderUdtForCustomerArr;
    BEGIN
        SELECT REF(s) BULK COLLECT INTO cosa
        FROM buy_order where ROWNUM<=1000 AND deref(buyer).pid = arg1;
        RETURN cosa;
    END;
END;
```

Implementando getOrderItems para el objeto Order

```
/* Creando arreglo orderItemForOrderArr en base al tipo orderItemUdt */
create type orderItemForOrderArr as VARRAY(1000) of REF orderItemUdt;
```

```
/* Alterando el tipo de dato orderUdt */
ALTER TYPE orderUdt ADD MEMBER FUNCTION getOrderItems(arg1 in Number) RETURN
orderItemForOrderArr CASCADE;
```

```
/* Implementando la función getOrderItems para el objeto Order */
CREATE OR REPLACE TYPE BODY orderUdt AS
    MEMBER FUNCTION getOrderItems(arg1 in Number) RETURN orderItemForOrderArr IS
        cosa orderItemForOrderArr;
    BEGIN
        SELECT REF(s) BULK COLLECT INTO cosa
        FROM order_item where ROWNUM<=1000 AND deref(relatedOrder).pid = arg1;
        RETURN cosa;
    END;
END;
```

Implementando getOrderItems para el objeto Item

```
/* Creando arreglo orderItemForItemArr en base al tipo orderItemUdt */
create type orderItemForItemArr as VARRAY(1000) of REF orderItemUdt;
```

```
/* Alterando el tipo de dato itemUdt */
ALTER TYPE itemUdt ADD MEMBER FUNCTION getOrderItems(arg1 in Number) RETURN
orderItemForItemArr CASCADE;
```



```
/* Implementando la función getOrderItems para el objeto Item */  
CREATE OR REPLACE TYPE BODY itemUdt AS  
    MEMBER FUNCTION getOrderItems(arg1 in Number) RETURN orderItemForItemArr IS  
        cosa orderItemForItemArr;  
    BEGIN  
        SELECT REF(s) BULK COLLECT INTO cosa  
        FROM order_item where ROWNUM<=1000 AND deref(relatedItem).pid = arg1;  
        RETURN cosa;  
    END;  
END;
```

2. Ejemplos de inserción de datos

2.1. Inserción de items

Para hacer lo anterior, se hace una inserción convencional, típica del modelo relacional. En el ejemplo, estamos insertando a 2 items.

```
INSERT INTO item VALUES (1, 6000, "Iphone", 5, NULL);
```

```
INSERT INTO item VALUES (2, 99.9, "USB", NULL);
```

2.2. Inserción de clientes

Para insertar un cliente, además del uso de la palabra clave **INSERT**, necesitaremos insertar el objeto de tipo **addressUdt**. Para ello, lo hacemos como si estuviéramos inicializando un objeto, como se muestra en el siguiente ejemplo:

```
INSERT INTO customer VALUES (1, 'Joshua', 'ftjoshua@gmail.com', addressUdt('Hacienda  
Catalunia', 4941, '12345'));
```

2.3. Poniendo al ítem de un OrderItem

Antes, crearemos un registro de tipo **OrderItem**

```
INSERT INTO order_item VALUES (1, 6000, 2, NULL, NULL);
```

Al ser una relación, podemos insertar el atributo *relatedItem* al objeto **OrderItem** mediante un **UPDATE** que lo iguale a un registro de su tabla de origen. Para poder hacer lo anterior, necesitamos seleccionar la referencia del objeto con la ayuda de la función reservada **REF**.

```
UPDATE order_item SET relatedItem = (SELECT REF(i) FROM item i WHERE i.PID=1) WHERE  
pid=1;
```

2.4. Creando órdenes

Para insertar una orden podemos usar la palabra clave **INSERT** sólo que, en lugar de usar la palabra clave **VALUES**, usamos una consulta. Además del id y el nombre de la orden, pasamos en la selección una referencia a un cliente con la ayuda de la función reservada **REF**.

```
INSERT INTO buy_order SELECT 1, 12000, REF(c) FROM customer c WHERE c.PID=1;
```

2.5. Asignando a una categoría un conjunto de productos

Previamente, crearemos una categoría cuyo atributo ítems sea nulo.

```
INSERT INTO category values (1, 'Gadgets', NULL);
```

En este caso, el atributo a modificar de category es *items*. Como en este caso estamos hablando de un **VARRAY**, tenemos que encerrar la selección que le igualaremos dentro de un constructor del tipo del atributo, es decir, ***itemUdtArr***.

```
UPDATE category SET items = itemUdtArr((SELECT REF(i) FROM item i where i.pid = 1), (SELECT  
REF(i) FROM item i where i.pid = 2) ) WHERE pid=1;
```

3. Consultas

Script de Consultas

Nota: Antes de probar estas consultas, se debe correr el script anexo a este manual, de nombre 'online_store.sql'.




Para hacer cualquier consulta en tablas de tipo objeto se puede usar la misma fórmula base del modelo relacional: **SELECT, FROM, WHERE**; adicionalmente, se deben usar ciertas funciones reservadas que veremos a continuación:

3.1. Mostrar las órdenes

Para ello necesitamos hacer una consulta a la tabla `buy_order`. Para poder ver los datos dentro de atributos que están definidos como objetos, necesitaremos hacer uso de la palabra clave **DEREF**. En este caso, sólo lo haremos sobre el atributo `buyer`. La palabra clave **AS** es para darle un "alias" a la columna que regresará la consulta.

```
/* Mostrando las órdenes*/  
SELECT pid, total, Deref(buyer).name AS customer_name FROM buy_order;
```

Corriendo el script mencionado al principio de esta sección, deberíamos ver algo así:

		PID		TOTAL		CUSTOMER_NAME
1		1		25998		Mateo
2		2		8833		Mateo
3		3		62995		Juan
4		4		1000		Andrea
5		5		30000		Mateo
6		6		22999.98		Irene
7		7		15333		Irene
8		8		73664		Andrea
9		9		1000		Juan
10		10		5333		Mateo

3.2. Mostrar la categoría a la que pertenece un item

Para esto necesitamos acceder al atributo `categories` de la tabla `item`, y posteriormente regresar el nombre de cada categoría. Para realizar esto existen varios enfoques; en estos momentos veremos 2:

1er enfoque: Podemos usar la palabra clave **THE** después del **FROM**. Esto nos permite acceder a los datos de un subquery que trae como resultado una relación a manera de registros individuales. De otra manera, sin el **THE** sólo obtendríamos un registro a modo de arreglo de tipo **categoryUdtArr**.

Posteriormente, hacemos uso de la palabra clave **COLUMN_VALUE** para acceder a cada registro, puesto que el resultado del subquery se encuentra en una columna que no tiene un nombre en específico. Con la palabra clave antes mencionada, podemos acceder a ese valor y, con la ayuda de **DEREF**, obtener los valores de nuestro objeto.

```
/* Haciendo la consulta con el 1er enfoque, ítem con pid = 2 */
SELECT Deref (COLUMN_VALUE).name
FROM THE (SELECT i.categories FROM item I WHERE i.pid = 2);
```

2do enfoque: La función reservada **TABLE** sirve para transformar el resultado de una consulta en una serie de registros. Haciendo esto, podemos ponerle un alias al resultado del **FROM** y, con ayuda de la palabra clave **COLUMN_VALUE**, obtener los valores deseados.

```
/* Haciendo la consulta con el segundo enfoque, ítem con pid = 2 */
SELECT m.COLUMN_VALUE .name
FROM TABLE(select i.categories
             from item i
             where i.pid = 2) m;
```

Los 2 enfoques dan el mismo resultado:

	COLUMN_VALUE.NAME
1	Fun
2	Business

3.3 Mostrar los ítems de una orden

Para esto usamos el **DEREF** en el **SELECT**. Nótese, a la vez ,que también podemos usar la instrucción **DEREF** en el **WHERE**.

```
/* Mostrando los ítems de la orden con pid = 1 */
SELECT Deref(relatedItem).name
FROM order_item oi
where Deref(relatedOrder).pid = 1;
```

	DEREF(RELATEDITEM).NAME
1	Kindle Fire
2	Ipad 2
3	Amazon Gift Card
4	Dell Vostro 2300

3.4 Items que no estén ligados a una categoría

Para poder hacer esta consulta haremos uso de un subquery en la cláusula **WHERE**. Para que un ítem pertenezca a 0 categorías, su atributo *categories* debe tener 0 en longitud. Para calcular esto

último, hacemos uso de la palabra clave **COUNT**. Para poder seleccionar el atributo *categories*, a su vez, debemos hacer uso de la palabra clave **THE** inmediatamente después del **FROM**.

```
SELECT i.name
FROM item i
WHERE (SELECT COUNT(deref(column_value).name) FROM
THE (SELECT item.categories
FROM item
WHERE item.PID = i.PID)) = 0;
```

1	NAME
1	Amazon Gift Card

3.5 Nombre del cliente en órdenes con más de 3 productos diferentes comprados

Para hacer esto, necesitaremos hacer uso de la cláusula **JOIN** para identificar el cliente de una orden. Posteriormente, tendremos que hacer un subquery en la cláusula **WHERE** para identificar la cantidad de orden ítems de cada orden, haciendo algo similar a lo que vimos en el query anterior. Nótese que podemos hacer también uso de la palabra clave **DEREF** para relacionar los atributos que determinan la unión.

```
SELECT c.name AS customer_name, bo.pid AS order_id
FROM customer c
JOIN buy_order bo
ON Deref(bo.buyer).pid = c.pid
WHERE (SELECT COUNT(*)
FROM order_item where Deref(relatedOrder).pid = bo.pid) > 3 ;
```

1	CUSTOMER_NAME	2	ORDER_ID
1	Mateo		1
2	Andrea		8

3.6 Órdenes de un cliente y con un ítem determinado

Para el status sólo debemos comparar el valor solicitado con el atributo *status* de la tabla estudiante. Para comparar el atributo *major* con el código otorgado, sólo hace falta usar la palabra clave **DEREF**. Finalmente, para poder obtener a los estudiantes de un club determinado, necesitamos hacer un subquery, donde el identificador de cada estudiante se encuentre dentro del atributo *members* del club otorgado.

```
/* Mostrando los ids de órdenes que haya hecho el cliente 'Andrea' del Iphone 4s*/
SELECT bo.pid AS order_id
FROM buy_order bo
WHERE Deref(buyer).name = 'Andrea'
AND 'Iphone 4s' IN (SELECT Deref(relatedItem).name
```

```
FROM order_item  
WHERE Deref(relatedOrder).pid = bo.pid);
```

	ORDER_ID
1	8

III. Fuentes de consulta

Creado de tipos:

http://download.oracle.com/docs/cd/B19306_01/server.102/b14200/statements_8001.htm

Creado de tablas:

http://download.oracle.com/docs/cd/B28359_01/server.111/b28286/statements_7002.htm

Creado de cuerpos de tipo:

http://download.oracle.com/docs/cd/B19306_01/server.102/b14200/statements_8002.htm#i2064997

Consultas de bases de datos objeto-relacional:

<http://infolab.stanford.edu/~ullman/fcdb/oracle/or-objects.html>